Trabalho Final – Paradigmas de Programação

Rafael Augusto Pissardo
RA: 11014910
Universidade Federal do ABC
rafael.pissardo@aluno.ufabc.edu.br

Victor Campagner de Barros
RA: 11024612
Universidade Federal do ABC
victor.c.barros@uol.com.br

INTRODUÇÃO

O projeto, desenvolvido com a linguagem Scheme, foi criado, pensado e planejado para a Disciplina de Paradigmas de Programação da Universidade Federal do ABC.

Inicialmente, o grupo pensou em desenvolver uma ferramenta de comunicação online que possibilitasse comunicação rápida e descomplicada aos usuários. A ídeia surgiu ao perceber que alguns serviços de *e-comerce* não possuem um serviço rápido e em tempo real em suas páginas que possibilite o dinamismo entre consumidor e loja.

Ao total foi gasto 2 meses de trabalho, entre idealização, programação da estrutura e ajustes finos. O projeto não se preocupa com a implementação Web do sistema, mas sim, com o desenvolvimento do conceito e o estudo da linguagem. Assim, o sistema de testes e domonstrações será feito utilizando comando do TelNet.

O projeto, como um tudo, tomou forma organicamente, e seu escopo foi levemente alterado conforme o grupo avançava na matéria e nas discuções. Portando, o projeto foi desenvolvido pensando em agilizar e facilitar a comunicação entre os dois extremos de uma cadeia utilizando a ferramenta desenvolvida em Scheme.

DEFINIÇÃO DO PROBLEMA

Algumas lojas de e-commerce têm um sistema de chat em suas páginas que possibilitam a comunicação rápida e dinâmica entre usuário e vendedor. Um exemplo é mostrado na figura a seguir:

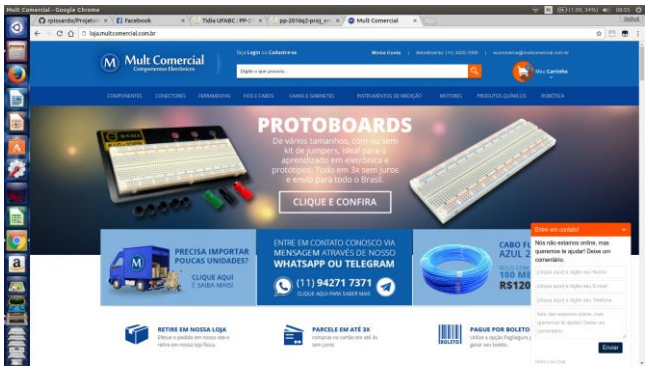

Imagem 1: Exemplo de Sistema

A ídeia fundamental do problema é criar uma ferramenta que torne dinâmico a comunicação, sem registros ou qualquer tipo de burocracia para ambos os lados do serviço.

ARQUITETURA E IMPLEMENTAÇÃO

Basicamente, tem-se dois lados:

1. Lado do serviço (chamemos de Servidor).
2. Lado do usuário (chamemos de Cliente).

O Servidor é responsável por criar um sistema de conexões que possibilite a troca de mensagens entre os Cliente. Ou seja, os

Clientes terão acesso ao bate-papo, se e somente se, o Servidor estiver online e com as configurações de portas corretas.

Ilustrativamente, temos as seguintes arquiteturas:

Imagem 2: Sistema de comunicação entre 2 clientes. Ariquitetura mais simples do sistema.

Logicamente, para o sistema ter sentido deve-se ter, no minimo, 2 clientes conectados ao mesmo servidor e na mesma porta. Porém, podemos ter conectado N clientes. Exemplo:

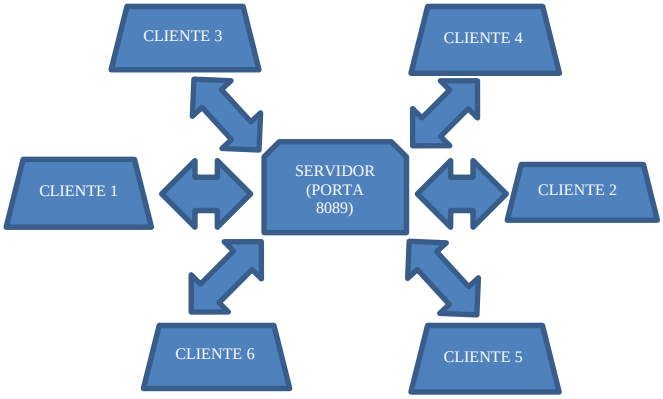

Imagem 3: Sistema de comunicação entre 6 clientes

AVALIAÇÃO

No início do projeto, o grupo pensou em ultizar alguma ferramenta de socket para scheme. Criou-se dois arquivos fundamentais, cliente.scm e server.scm. O cliente.scm era resposável por capturar as mensagens digitadas pelo usuário e enviar, via socket, ao server.scm (que estaria rodando na máquina do segundo usuário) através de um socket. Pois bem, os testes iniciais foram muito satisfatórios, conforme ilustra-se na figura a seguir.

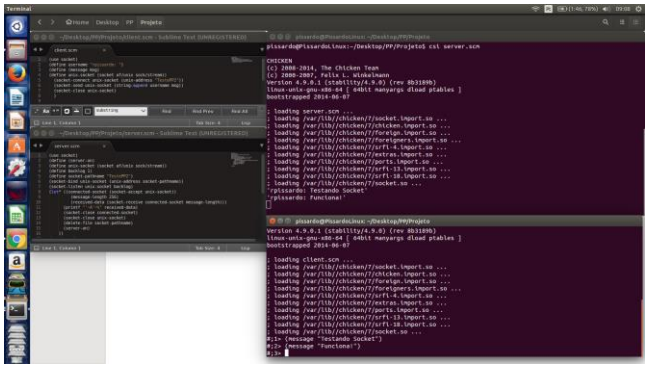

Imagem 4: Sistema escrito com socket

Porém, um problema surgiu. Como executar o server.scm junto com o cliente.scm utlizando concorrência? Afinal, o server.scm "trava" o código como um todo e não permite que nenhuma outra operção seja realizada. O grupo levou grande parte do tempo do projeto tentando resolver tal problema, porém sem sucesso. Todas as formas de contornar o bloqueio do socket de forma que a mensagem enviada e a mensagem recebida pudessem ser processadas no mesmo terminal, foram fracassadas.

O sistema não era totalmente falho. Porém, não era o que o grupo buscava. O sistema funcionava quando aberto em 4 terminais. Desses 4, 2 ficam responsáveis por ouvir o socket e outros dois por enviar as mensagens.

Na imagem abaixo, ilustramos como é o sistema para dois clientes. Lembrando que os sockets deveriam ter uma forma em cruz, ou seja, o servidor 1 deveria conectar em socket com o cliente 2, e o servidor 2 em socket com o cliente 1.

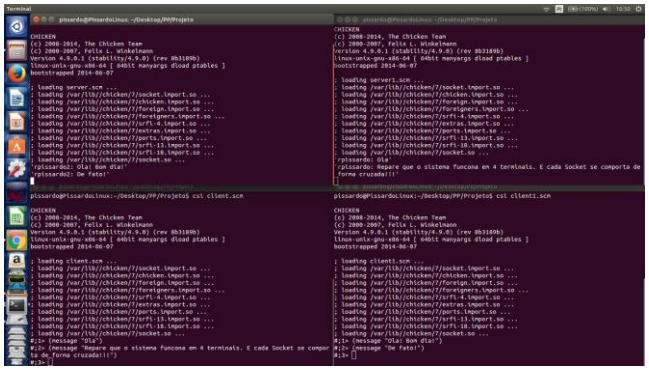

Imagem 5: Sistema cruzado com Socket

Além dos problemas citados, acima, este sistema em socket não possibilita a conexão em cruz com mais de 2 clientes.

Ou seja, não poderíamos conectar uma "sala" com mais de 2 clientes. Caso o sistema fosse com 3 clientes, deveríamos abrir 8 terminais (dois terminais para cada conexão com cada cliente). Ou seja o número de terminais abertos no sistema seria de $2^{número\_de\_clientes}$.

A partir de uma aula de Comunição e Redes, o grupo teve a ídeia de realizar testes utilizando a ferramenta TelNet. Nela, o sistema abre porta especificas do computador e permite que qualquer pessoa envie dados à essas portas. Bastava, então desenvolver um sistema em Scheme que ouvisse estas portas e direcionasse os conteúdos recebidos ao Clientes.

O sistema implementado foi um sucesso. Em um terminal de qualquer máquina, uma porta é disponibilizada e o sistema iniciado executando o comando (chat 8085), sendo que 8085 é a porta que o Servidor está disponibilizando àquela sala.

Agora, com o Servidor liberado, cada Cliente deve entrar com o sistema TelNet digitando em seus terminais: telnet IP_do_Servidor Porta_Disponibilizada. Pronto, o Cliente está conectado com todos os outros Cliente que estão utilizando a mesma porta. O exemplo abaixo, mostra a comunição entre terminais da mesma máquina, por isso, utiliza-se localhost como IP.

O principal objetivo do grupo foi concluído, restava, agora, implementar concorrência e threads ao sistema, bem como criar algumas funções úteis ao Cliente. Na imagem 6, algumas ferramentas ja estão implementadas. Como listar usuários e sair da sala.

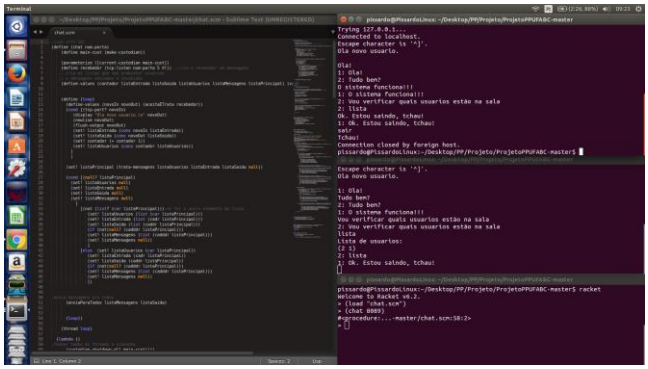

Imagem 6: Sistema com telnet funcionando. Algumas ferramentas de usuários implementadas.

Outra ferramenta importante fornecida ao usuário é a possibilidade de modificar seu nome de usuário inicial. Ao entrar no sistema, o usuário recebe um ID que é referente a sua posicão de entrada na sala. Ou seja, se o usuário é o primeiro a entrar, seu ID será 1, o segundo terá ID 2 e assim por diante. Porém ao digitar "usuario nome" o usuário tem seu ID substituido pelo "nome".

Veja a imagem 7.

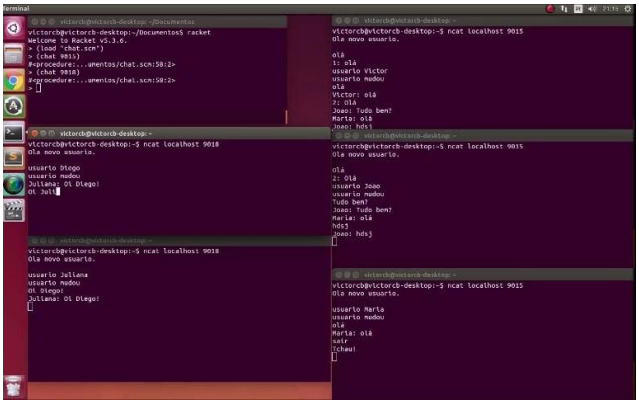

Imagem 7: Usuários com nomes alterados.

Na imagem 7, tem-se um exemplo claro da função para mudar o nome do usuário.

Outra função simples que não requer muitas explicações, tão pouco imagens, é a função sair. Ao digitar "sair", o usuário é desligado imediatamente do sistema e nao receberá mais mensagens.

DISCUSSÕES

O grupo levou muito tempo para perceber que a utilização do socket na forma como esta programando, não seria uma opção viável. Bem como gastou boa parte do tempo definindo o escopo real do projeto. O sistema, ou melhor, o conceito do sistema pensado, possibilita uma série de ramificações e caminhos possíveis. Com isso, é natural que o foco fique um pouco abalado.

Entretanto e concluindo, o grupo mostrou-se muito capaz e flexível durante a realização do projeto. Ao perceber que não daria, de fato, para usar socket, e com a ideia de usar o TelNet, o grupo teve três dias para reorganizar as ídeias e desenvolver um novo código, praticamente do ínicio.

Ambos do grupo trabalharam de forma árdua. O código era atualizado e trocado via email ou mensagens de Facebook. No GitHub, foram adicionados apenas os códigos com resultados satisfatórios ou julgados interessantes pelo grupo. Devido ao tempo corrido de ambos os integrantes, os códigos eram escritos e resivados de acordo com a disponibilidade de tempo individual.

REFERÊNCIAS

[1] SEBESTA, Robert W. **Conceitos de Linguagens de Programação**, 9 ed. Porto Alegre, RS: Bookman, 2011.

[2] GOERZEN, John. **Foundations of Python Network Programming**: The comprehensive guide to build network applications with Python. Second Edition, Online Resource. Disponível em: http://dx.doi.org/10.1007/978-1-4302-3004-5. Acesso em: 21 Ago. 2016.

[3] TANENBAUM, Andrew Stuart; ANDREW TANENBAUM. **Computer Networks**, 4 ed. Upper Saddle River, USA: Prentice Hall.

Link do Projeto para o GitHub: github.com/rpissardo/ProjetoPPUFABC